This listing of claims will replace all prior versions, and listings, of claims in the application.

Listing of Claims:

1. (Previously Presented) A method of translating compiled programming code from a first compiled code state to a second compiled code state, the programming code in the first compiled code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, the method comprising the steps of:

identifying the plurality of basic blocks in the first compiled code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, and without information from a source external to the first compiled code state of the programming code, all identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the identified branch table; and

translating the programming code from the first compiled code state to the second compiled code state based at least in part on the updated CFG.

2. (Original) The method of claim 1 wherein the exploring step comprises the steps of:

for each explored basic block, constructing a corresponding code graph / representation (code graph) of the instructions in such basic block; and

traversing each code graph to determine the set of destination addresses from the dynamic branch.

3. (Original) The method of claim 2 wherein each code graph is rooted directed acyclic graph having interconnected nodes, each node being one of:

an instruction node representing an instruction in the corresponding basic block;

an argument node representing an argument in the corresponding basic block;

an apply node edging to an instruction node and to an argument node and representing the application of such argument node to such instruction node, the apply node in certain instances also being an argument node edged to by another node;

a stack node edging to a pair of argument nodes and acting as an argument node having the pair of argument nodes;

a missing argument node representing a missing argument supplied from a different basic block; and

an alias node edged to by a stack node or apply node and edging to an argument remote from such stack node, and representing such remote argument to such stack node.

Claims 4-13 (Canceled).

14. (Previously Presented) A translator operating on a processor for translating compiled programming code from a first compiled code state to a second compiled code state, the programming code in the first compiled code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, the translator:

identifying the plurality of basic blocks in the first compiled code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, and without information from a source external to the first compiled code state of the programming code, all identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine, a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the identified branch table; and

translating the programming code from the first compiled code state to the second compiled code state based at least in part on the updated CFG.

15. (Original) The translator of claim 14 wherein the translator:

for each explored basic block, constructs a corresponding code graph / representation (code graph) of the instructions in such basic block; and

traverses each code graph to determine the set of destination addresses from the dynamic branch.

16. (Original) The translator of claim 15 wherein each code graph is a rooted directed acyclic graph having interconnected nodes, each node being one of:

an instruction node representing an instruction in the corresponding basic block;

an argument node representing an argument in the corresponding basic block;

an apply node edging to an instruction node and to an argument node and representing the application of such argument node to such instruction node, the apply node in certain instances also being an argument node edged to by another node;

a stack node edging to a pair of argument nodes and acting as an argument node having the pair of argument nodes;

a missing argument node representing a missing argument supplied from a different basic block; and

an alias node edged to by a stack node or apply node and edging to an argument remote from such stack node, and representing such remote argument to such stack node.

Claims 17-26 (Canceled).

27. (Previously Presented) In connection with translating compiled programming code from a first compiled code state to a second compiled code state, the programming code in the first compiled code state comprising a plurality of basic blocks, each basic block comprising a set of instructions, at least one basic block ending in a dynamic branch, the dynamic branch being a transfer to one of a set of destinations based on a calculation of a destination address, a computer- readable medium having computer-executable instructions for performing steps comprising:

identifying the plurality of basic blocks in the first compiled code state of the programming code;

identifying links between the identified basic blocks;

constructing a control flow graph / representation (CFG) of the programming code based on the identified basic blocks and identified links, the CFG being in a preliminary form;

identifying at least one basic block ending in a dynamic branch;

exploring, based on the CFG, and without information from a source external to the first compiled code state of the programming code, all identified basic blocks that lead to the dynamic branch as far back as is necessary to fully determine a set of destination addresses for the dynamic branch, the set of destination addresses defining the set of destinations from the dynamic branch;

examining the set of destinations to identify a branch table;

updating the CFG to reflect the set of destinations and the identified branch table; and

translating the programming code from the first compiled code state to the second compiled code state based at least in part on the updated CFG.

28. (Original) The computer-readable medium of claim 27 wherein the exploring step comprises the steps of:

for each explored basic block, constructing a corresponding code graph / representation (code graph) of the instructions in such basic block; and

traversing each code graph to determine the set of destination addresses from the dynamic branch.

29. (Original) The computer-readable medium of claim 28 wherein each code graph is a rooted directed acyclic graph having interconnected nodes, each node being one of:

an instruction node representing an instruction in the corresponding basic block;

an argument node representing an argument in the corresponding basic block;

an apply node edging to an instruction node and to an argument node and representing the application of such argument node to such instruction node, the apply node in certain instances also being an argument node edged to by another node;

a stack node edging to a pair of argument nodes and acting as an argument node having the pair of argument nodes;

a missing argument node representing a missing argument supplied from a different basic block; and

an alias node edged to by a stack node or apply node and edging to an argument remote from such stack node, and representing such remote argument to such stack node.

Claims 30-39 (Canceled).